Decision Trees: From Root to Leaf in Machine Learning

Ahmed BADI
ahmedbadi905@gmail.com
linkedin.com/in/badi-ahmed

December 5, 2025

Abstract

Decision trees are one of the most intuitive and widely used machine learning algorithms for both classification and regression tasks. This article provides a comprehensive exploration of decision tree methodology, covering the fundamental concepts of tree structure, splitting criteria (entropy, Gini impurity, information gain), and the recursive partitioning process. We examine the mathematical foundations underlying tree construction, discuss practical considerations such as pruning techniques to prevent overfitting, and analyze the advantages and limitations of this approach. While individual decision trees have limitations—particularly their tendency to overfit and high variance—they form the foundation of powerful ensemble methods like Random Forests and Gradient Boosting Machines that consistently achieve state-of-the-art performance [1], [2], [3].

The article also explores important variants including CART (Classification and Regression Trees), ID3, and C4.5 algorithms, while highlighting real-world applications across domains such as medical diagnosis, credit risk assessment, fraud detection, and customer segmentation [4], [5], [6], [7]. Through clear explanations, mathematical rigor, and visual examples, this guide serves both newcomers seeking to understand the basics and practitioners looking to deepen their knowledge of decision tree algorithms.

As you apply decision trees in your work, remember that the algorithm's transparency is both its greatest strength and a responsibility. The decisions your tree makes should not only be accurate but also fair, explainable, and aligned with domain knowledge. Start with simple, shallow trees for interpretability, and gradually increase complexity or move to ensembles when performance demands it.

Keywords: Decision Trees, Classification, Regression, Entropy, Gini Impurity, Information Gain, Pruning, CART, ID3, C4.5, Machine Learning, Interpretable Models.

1 Introduction

Imagine you are a doctor trying to diagnose whether a patient has the flu. You might ask: "Does the patient have a fever?" If yes, you ask: "Does the patient have a cough?" If yes again, you might conclude: "It is likely the flu." This simple flowchart of questions is exactly how a decision tree works.

Decision trees are among the most natural and interpretable machine learning algorithms [2], [3], [4]. Unlike many black-box models, they mirror human decision-making by breaking down complex problems into a series of simple, yes-or-no questions. This makes them powerful for both technical and non-technical audiences.

What makes decision trees special? First, they are transparent: you can literally see the decision process by following the tree from root to leaf. Second, they handle both numerical and categorical data with minimal preprocessing. Third, they automatically perform feature selection by choosing the most informative variables at each split. Fourth, they can capture non-linear relationships and interactions between features without explicitly specifying them [1].

Decision trees have been successfully applied in credit scoring, medical diagnosis, marketing, manufacturing, and many other areas. They also form the backbone of popular ensemble methods such as Random Forests and Gradient Boosting Machines, which dominate many machine learning competitions [6], [7].

2 What is a Decision Tree?

2.1 Tree Structure

A decision tree is a hierarchical structure that makes predictions by recursively splitting data into subsets based on feature values. Think of it as an upside-down tree that grows from top to bottom.

The tree consists of:

- **Root node**: represents the entire dataset.
- **Internal (decision) nodes**: contain a test on one feature (e.g., Income $\leq$ 50,000).
- **Leaf (terminal) nodes**: represent final predictions (class labels or numeric values).
- **Branches (edges)**: connections between nodes, corresponding to outcomes of tests.
- **Depth**: length of the longest path from root to leaf; deeper trees model more complex patterns but risk overfitting.

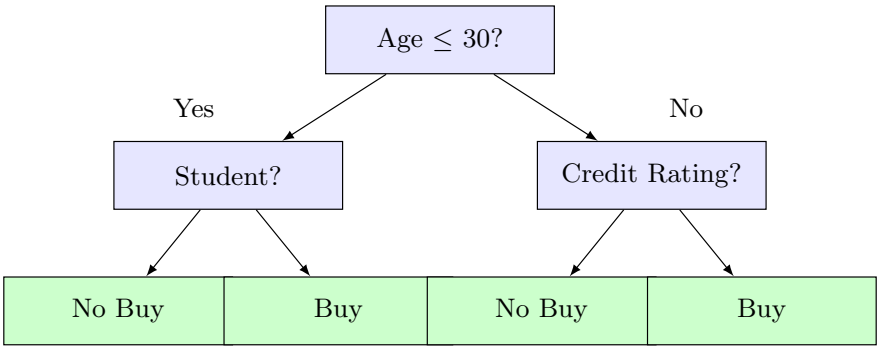

Figure 1: Example of a simple decision tree for predicting customer purchase behavior.

2.2 Predictions for Classification and Regression

For **classification**, a decision tree predicts the class by:

1. Starting at the root node.
2. Evaluating the test condition at the current node.
3. Following the branch corresponding to the test outcome.
4. Repeating until reaching a leaf node.
5. Returning the majority class at that leaf.

For **regression**, the procedure is the same, but each leaf contains a numerical prediction, typically the mean of target values of the training samples that reached that leaf [1], [8].

3 Building a Decision Tree

3.1 Recursive Partitioning

Trees are built by a greedy, top-down process known as *recursive binary splitting* [2]:

1. Start with all training data at the root node.
2. For each candidate feature and split point, compute how much the split improves impurity (purity of child nodes).
3. Select the feature and split that give the largest impurity reduction.
4. Partition the data into child nodes and repeat the process recursively.
5. Stop when a stopping criterion is met (maximum depth, minimum number of samples per node, or zero impurity).

This process is greedy: it makes the best local choice at each step, without exploring all possible trees (which would be computationally intractable).

3.2 Impurity Measures for Classification

3.2.1 Entropy and Information Gain

Entropy measures the impurity of a node based on class proportions. For a node containing samples from C classes with proportions $p_1, \ldots, p_C$, entropy is:

$$H(S) = -\sum_{i=1}^{C} p_i \log_2(p_i). \tag{1}$$

Entropy is 0 when the node is perfectly pure (all samples in one class) and maximal when classes are evenly mixed.

When splitting a node S on feature A into subsets S_v, the **information gain** is:

$$IG(S, A) = H(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} H(S_v), \tag{2}$$

where $|S|$ is the number of samples in the node. ID3 and C4.5 use entropy-based criteria [4], [5].

3.2.2 Gini Impurity

CART uses the **Gini impurity** as default [2]. For class proportions $p_1, \ldots, p_C$, it is:

$$G(S) = 1 - \sum_{i=1}^{C} p_i^2. \tag{3}$$

Gini is also 0 for pure nodes and larger when classes are mixed.

The Gini decrease for a split is:

$$\Delta G(S, A) = G(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} G(S_v). \tag{4}$$

3.2.3 Misclassification Error

A simpler impurity measure is the misclassification error:

$$E(S) = 1 - \max_i(p_i), \tag{5}$$

the fraction of samples that do not belong to the majority class. It is intuitive but less sensitive to changes in class proportions, so it is typically used for pruning rather than splitting.

3.2.4 Visual Comparison of Impurity Measures

Figure 2 shows entropy, Gini impurity, and misclassification error for a binary node as the proportion p of class 1 varies.

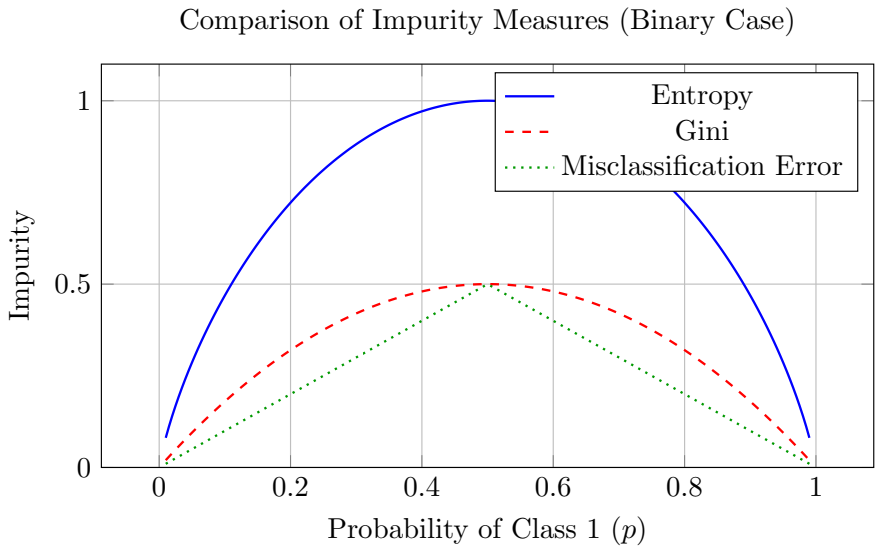

Figure 2: Entropy, Gini impurity, and misclassification error as functions of class proportion p.

3.3 Variance Reduction for Regression Trees

For regression trees, impurity is measured by variance. For a node S with targets y_i, the variance is:

$$Var(S) = \frac{1}{|S|}\sum_{i \in S}(y_i - \bar{y})^2, \tag{6}$$

where $\bar{y}$ is the mean of y_i in S.

The variance reduction for a split on feature A is:

$$\Delta Var(S, A) = Var(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} Var(S_v). \tag{7}$$

CART uses this criterion for regression trees [1], [2].

4 Key Decision Tree Algorithms

4.1 ID3

ID3 (Iterative Dichotomiser 3) was one of the first popular decision tree algorithms [4]. Its main characteristics:

- Uses information gain (entropy reduction) as splitting criterion.
- Handles categorical features; continuous features must be discretized.
- Produces multi-way splits (one branch per category).
- No pruning; tends to overfit.

4.2 C4.5

C4.5 is an extension of ID3 that addresses several limitations [5]:

- Uses **gain ratio**, which normalizes information gain:

$$GainRatio(S, A) = \frac{IG(S, A)}{SplitInfo(S, A)}, \tag{8}$$

with

$$SplitInfo(S, A) = -\sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \log_2 \left(\frac{|S_v|}{|S|} \right). \tag{9}$$

- Handles continuous features by dynamically choosing thresholds.
- Deals with missing values by fractional assignment.
- Includes post-pruning to reduce overfitting.

4.3 CART

CART (Classification and Regression Trees) is one of the most widely used algorithms [2]. Key properties:

- Uses Gini impurity for classification and variance reduction for regression.
- Produces only **binary** splits.
- Supports numerical and categorical variables.
- Uses **cost–complexity pruning** to control tree size.
- Forms the basis for Random Forests and many implementations in libraries like scikit-learn [8].

4.4 Algorithm Comparison

Table 1: High-level comparison of ID3, C4.5, and CART.

Algorithm	**Split Criterion**	**Splits**	**Pruning**
ID3	Information Gain (entropy)	Multi-way	None (stopping rules only)
C4.5	Gain Ratio	Multi-way	Error-based post-pruning
CART	Gini (class), MSE (reg)	Binary	Cost–complexity pruning

5 Overfitting and Pruning

5.1 Overfitting in Decision Trees

Deep trees can fit training data extremely well, even memorizing noise, which leads to poor generalization on new data [1], [2]. Symptoms:

- Training accuracy very high, test accuracy significantly lower.
- Many levels (large depth) and leaves with very few samples.
- Highly specific, non-intuitive rules.

5.2 Pre-Pruning (Early Stopping)

Pre-pruning stops tree growth before it perfectly fits the data. Common constraints:

- Maximum depth (`max_depth`).
- Minimum samples required to split a node (`min_samples_split`).
- Minimum samples per leaf (`min_samples_leaf`).
- Minimum impurity decrease needed for a split.

Pre-pruning is simple and fast, but it can sometimes stop too early, missing useful structure.

5.3 Post-Pruning

Post-pruning grows a large tree and then removes branches that do not improve validation performance.

5.3.1 Cost–Complexity Pruning (CART)

CART introduces a complexity-penalized objective:

$$R_\alpha(T) = R(T) + \alpha|T|, \tag{10}$$

where:

- $R(T)$ is the training error (e.g., misclassification rate) of tree T,
- $|T|$ is the number of leaf nodes,
- $\alpha \geq 0$ controls the penalty for complexity.

For each α, the algorithm finds the subtree T_α that minimizes $R_\alpha(T)$, generating a sequence of nested subtrees. Cross-validation is then used to select the best α [2].

5.3.2 Reduced Error Pruning

A simpler approach:

1. Split data into training and validation sets.
2. Train a full tree on the training set.
3. For each internal node, consider replacing the subtree by a leaf.
4. If validation error does not increase, prune the subtree.
5. Repeat until no further pruning improves validation performance.

6 Handling Different Feature Types

6.1 Numerical Features

For a numerical feature x, CART typically:

1. Sorts the unique values of x.
2. Considers midpoints between consecutive values as candidate thresholds t.
3. Evaluates impurity reduction for splits of the form $x \leq t$ vs $x > t$.
4. Chooses the threshold with the maximum gain.

6.2 Categorical Features

For categorical features with values $\{v_1, \ldots, v_k\}$:

- ID3/C4.5: often use multi-way splits, one branch per category.
- CART: searches binary partitions of the category set (in theory $2^{k-1} - 1$ possibilities, often approximated).

Ordered categorical features (e.g., low/medium/high) can be treated as numeric with appropriate coding.

7 Feature Importance

Decision trees naturally provide measures of feature importance based on impurity reduction [2], [6].

The unnormalized importance of feature f is:

$$Importance(f) = \sum_{t \in \text{nodes using } f} \Delta I_t \, n_t, \tag{11}$$

where ΔI_t is impurity decrease at node t and n_t is the number of samples in that node. Normalizing:

$$NormalizedImportance(f) = \frac{Importance(f)}{\sum_{f'} Importance(f')}. \tag{12}$$

This helps answer:

- Which features drive predictions the most?
- Are some features redundant or uninformative?
- Which variables are worth measuring or improving?

8 Decision Trees vs Other Models

Table 2: Decision trees compared to other common machine learning models.

Model	**Advantages over Trees**	**Disadvantages vs Trees**
Logistic Regression	More stable, good for linear relationships, well-calibrated probabilities [1]	Cannot easily model complex non-linear interactions; less intuitive than small trees
k-Nearest Neighbors	Simple, no training; flexible local decision boundaries [1]	Slow at prediction; requires feature scaling; hard to explain decisions
Support Vector Machines	Effective in high dimensions; powerful with kernels [1]	Black-box; feature scaling required; hyperparameter tuning can be difficult
Neural Networks	Can learn highly complex patterns; excellent for images, text, signals [9]	Require large datasets; many hyperparameters; poor interpretability
Random Forest / Gradient Boosting	Higher accuracy and stability; reduce variance and overfitting [6], [7]	Ensembles of many trees are harder to interpret; more computationally expensive

In practice, a single decision tree is often used as:

- An interpretable baseline.
- A "white-box" explanation model alongside more complex ensembles.

9 Practical Considerations and Hyperparameters

Key hyperparameters for decision trees (e.g., in scikit-learn) include [8]:

- **max_depth**: maximum depth of the tree.
- **min_samples_split**: minimum samples needed to split a node.
- **min_samples_leaf**: minimum samples per leaf.
- **max_features**: number of features to consider when looking for the best split.
- **criterion**: "gini" or "entropy" (for classification).
- **class_weight**: adjusting for imbalanced classes.

These should be tuned with cross-validation to balance bias and variance.

For imbalanced data:

- Use `class_weight="balanced"` or custom weights.
- Apply resampling (oversampling minority or undersampling majority).
- Evaluate with precision, recall, F1-score, and ROC–AUC rather than accuracy alone.

10 Conclusion

Decision trees remain a core tool in machine learning because they combine:

- Human-friendly interpretability.
- Ability to handle mixed data types with minimal preprocessing.
- Natural handling of non-linear relationships and feature interactions.

In this article, we:

- Defined the structure and prediction mechanism of decision trees.
- Explained splitting criteria (entropy, Gini, variance reduction).
- Presented key algorithms (ID3, C4.5, CART) and pruning strategies.
- Discussed strengths, weaknesses, and comparisons with other models.
- Highlighted real-world applications and practical tips.

As a rule of thumb, start with a small, shallow tree to understand the problem and communicate insights. When higher predictive performance is needed, move to ensembles such as Random Forests and Gradient Boosted Trees, which are built on top of the same fundamental idea: splitting data into meaningful, interpretable regions using decision rules.

References

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd. New York: Springer, 2009, ISBN: 978-0-387-84857-0.

[2] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA: Wadsworth, 1984, ISBN: 0-534-98053-8.

[3] L. Rokach and O. Maimon, *Data Mining with Decision Trees: Theory and Applications*. World Scientific, 2008, ISBN: 978-9812771711.

[4] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986. DOI: 10.1007/BF00116251.

[5] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993, ISBN: 1-55860-238-0.

[6] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. DOI: 10.1023/A:1010933404324.

[7] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001. DOI: 10.1214/aos/1013203451.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, et al., *Scikit-learn: Machine learning in Python*, *Journal of Machine Learning Research*, 12:2825–2830, Decision tree implementation documentation: https://scikit-learn.org/stable/modules/tree.html, 2011.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016, ISBN: 978-0262035613.